OPERATING SYSTEM CONCEPTS

Chapter 5. CPU Scheduling (Contd.)

A/Prof. Kai Dong

dk@seu.edu.cn
School of Computer Science and Engineering,
Southeast University

April 12, 2024

Contents

東南大學
SOUTHEAST UNIVERSITY

Contents

Lottery Scheduling
Proportional Share

- Suppose a virtualized data center, where
  - You might like to assign one-quarter of your CPU cycles to the Windows VM and the rest to your base Linux installation
- Proportional-share scheduler (fair-share scheduler)
  - Instead of optimizing for turnaround or response time, a scheduler might instead try to guarantee that each job obtain a certain percentage of CPU time.

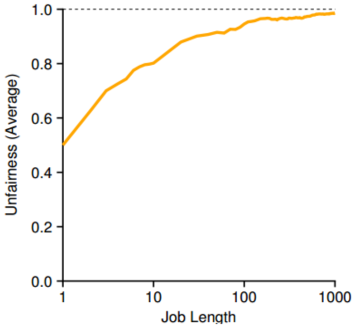
Lottery Scheduling

- Every so often, hold a lottery to determine which process should get to run next;
- Processes that should run more often should be given more chances to win the lottery.
- Tickets are used to represent the share of a resource that a process (or user or whatever) should receive.
    - Ticket currency — Currency allows a user with a set of tickets to allocate tickets among their own jobs in whatever currency they would like; the system then automatically converts said currency into the correct global value.
    - Ticket transfer — A process can temporarily hand off its tickets to another process.
    - Ticket inflation — A process can temporarily raise or lower the number of tickets it owns.

Lottery Scheduling
An Unfairness Metric

- Suppose two jobs competing against one another, each with the same number of tickets and same run time.

- An unfairness metric U:
    - The time the first job completes divided by the time that the second job completes.
    - With a perfect fair scheduler, two jobs should finish at roughly the same time, i.e., U=1.



Lottery Scheduling
Probabilistic Vs. Deterministic

- Lottery scheduling is probabilistic.
    - Only as the jobs run for a significant number of time slices does the lottery scheduler approach the desired fairness.
- Stride scheduling, is a deterministic fair-share scheduler.
    - Each job in the system has a stride, which is inverse in proportion to the number of tickets it has.
    - Every time a process runs, we will increment a counter for it (called its pass value) by its stride to track its global progress.
    - At any given time, pick the process to run that has the lowest pass value so far.

Lottery Scheduling
Stride Scheduling

- Suppose three processes (A, B and C), with stride values of 100, 200 and 40, and all with pass values initially at 0.

Pass(A) (stride=100)	Pass(B) (stride=200)	Pass(C) (stride=40)	**Who Runs?**
0	0	0	A
100	0	0	B
100	200	0	C
100	200	40	C
100	200	80	C
100	200	120	A
200	200	120	C
200	200	160	C
200	200	200	…

- **Why using probabilistic, not deterministic?**

Lottery Scheduling
Why Not Deterministic?

- Lottery scheduling has one nice property that stride scheduling does not: no global state.
    - Imagine a new job enters in the middle of our stride scheduling example above; what should its pass value be? Should it be set to 0? If so, it will monopolize the CPU.

Contents

Thread Scheduling

- Distinction between user-level and kernel-level threads
- On operating systems that support threads, it is kernel-level threads — not processes — that are being scheduled
  - Kernel thread scheduled onto available CPU is system-contention scope (SCS) — competition among all threads in system
- Many-to-one and many-to-many models, thread library schedules user-level threads to run on LWP
  - Typically done via priority set by programmer
  - Known as process-contention scope (PCS) since scheduling competition is within the process

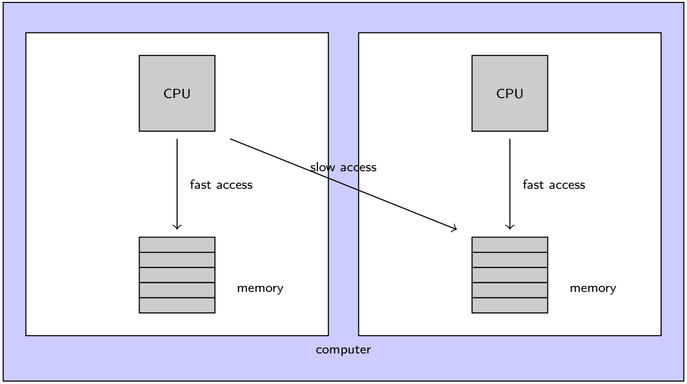
Contents

Multiple-Processor Scheduling
Cache

- Caches are based on the notion of locality
  - Temporal locality: when a piece of data is accessed, it is likely to be accessed again in the near future
  - Spatial locality: if a program accesses a data item at address x, it is likely to access data items near x as well
- Caches in multi-processor Architecture
  - Cache Affinity (a.k.a., Processor Affinity) — A process, when run on a particular CPU, builds up a fair bit of state in the caches of the CPU. The next time the process runs, it is often advantageous to run it on the same CPU, as it will run faster if some of its state is already present in the caches on that CPU

Multiple-Processor Scheduling
Cache Affinity

- Memory-placement algorithms can also consider affinity.



Multiple-Processor Scheduling
Asymmetric Multiprocessing

- **Asymmetric Multiprocessing** — single-queue multiprocessor scheduling (SQMS)
- Simply reuse the basic framework for single processor scheduling, by putting all jobs that need to be scheduled into a single queue
- Cache affinity problem

$$Queue \rightarrow A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow NULL$$

CPU_0	A	E	D	C	B
CPU_1	B	A	E	D	C
CPU_2	C	B	A	E	D
CPU_3	D	C	B	A	E

CPU_0	A	E	A		
CPU_1	B		E	B	
CPU_2	C			E	C
CPU_3	D				E

Multiple-Processor Scheduling
Symmetric Multiprocessing

- **Symmetric Multiprocessing** — multi-queue multiprocessor scheduling (MQMS)
- One queue per CPU. Each queue will likely follow a particular scheduling discipline, such as round robin.

$Queue_0 \rightarrow A \rightarrow C \rightarrow NULL$ $Queue_1 \rightarrow B \rightarrow D \rightarrow NULL$

CPU_0	A	C	A	C	A	C	B
CPU_1	B	D	B	D	B	D	

- Load imbalance problem.
  - Migration — By migrating a job from one CPU to another, true load balance can be achieved.

Contents

Real-Time CPU Scheduling

- **Soft real-time systems** — no guarantee as to when critical real-time process will be scheduled
- **Hard real-time systems** — task must be serviced by its deadline
- Two types of latencies affect performance
    - Interrupt latency — time from arrival of interrupt to start of routine that services interrupt
    - Dispatch latency — time for scheduling dispatcher to take current process off CPU and switch to another
- Conflict phase of dispatch latency:
    - Preemption of any process running in kernel mode
    - Release by low-priority process of resources needed by high-priority processes

Real-Time CPU Scheduling

Event				
Application response time	Interrupt response	Interrupt latency		Processor instruction or system in critical region; locks out interrupt.
				System saves or restores registers, and vectors to interrupt routine.
		Interrupt processing		Driver's interrupt routine sends message to wake up sleeping process.
				Returns process interrupt.
		Dispatch latency	Wakeup	Low-priority processes release resources or provide input to higher priority process.
			Dispatch	Reschedules to run highest priority task.
		Priority task		Calculates response.
Response to event				